NYU, Tandon School of Engineering
Extended Bridge to CS — Summer 2018

Homework #9
Due by Friday 9/14 11:55pm

Submission instructions:

1. Pay special attention to the style of your code. Indent your code correctly, choose meaningful names for your variables, define constants where needed, choose most suitable control statements, break down your solutions by defining functions, etc.
2. You should submit your homework in the NYU Classes system.
3. For this assignment, you should turn in 4 '.cpp' files.
   Name these files: 'YourNetID_hw9_q1.cpp', 'YourNetID_hw9_q2.cpp', etc.

Question 1:

Write a program that will read in a line of text and output the number of words in the line and the number of occurrences of each letter.

Define a word to be any string of letters that is delimited at each end by either whitespace, a period, a comma, or the beginning or end of the line.

You can assume that the input consists entirely of letters, whitespace, commas, and periods.

When outputting the number of letters that occur in a line, be sure to count upper and lowercase versions of a letter as the same letter.

Output the letters in alphabetical order and list only those letters that do occur in the input line.

Your program should interact with the user **exactly** as it shows in the following example:

```
Please enter a line of text:
I say Hi.
3       words
1       a
1       h
2       i
1       s
1       y
```

Notes:

1. Think how to break down your implementation to functions.
2. Pay attention to the running time of your program. If the input line contains n characters, an efficient implementation would run in a linear time (that is $\Theta(n)$).

Question 2:

Two strings are **anagrams** if the letters can be rearranged to form each other. For example, "Eleven plus two" is an anagram of "Twelve plus one". Each string contains one 'v', three 'e's, two 'l's, etc.

Write a program that determines if two strings are anagrams. The program should not be case sensitive and should disregard any punctuation or spaces.

Notes:

1. Think how to break down your implementation to functions.
2. Pay attention to the running time of your program. If each input string contains n characters, an efficient implementation would run in a linear time (that is $\Theta(n)$).

Question 3:

In this question, you will write **four versions** of a function `getPosNums` that gets an array of integers `arr`, and its logical size. When called it creates a new array containing only the positive numbers from `arr`.
For example, if `arr=[3, -1, -3, 0, 6, 4]`, the functions should create an array containing the following 3 elements: `[3, 6, 4]`,

The four versions you should implement differ in the way the output is returned.
The prototypes of the functions are:

a) `int* getPosNums1(int* arr, int arrSize, int& outPosArrSize)`
returns the base address of the array (containing the positive numbers), and updates the output parameter `outPosArrSize` with the array's logical size.

b) `void getPosNums2(int* arr, int arrSize,`
`int*& outPosArr, int& outPosArrSize)`
updates the output parameter `outPosArr` with the base address of the array (containing the positive numbers), and the output parameter `outPosArrSize` with the array's logical size.

c) `int* getPosNums3(int* arr, int arrSize, int* outPosArrSizePtr)`
returns the base address of the array (containing the positive numbers), and uses the pointer `outPosArrSizePtr` to update the array's logical size.

d) `void getPosNums4(int* arr, int arrSize,`
`int** outPosArrPtr, int* outPosArrSizePtr)`
uses the pointer `outPosArrPtr` to update the base address of the array (containing the positive numbers), and the pointer `outPosArrSizePtr` to update the array's logical size.

Note: You should also write a `main` program that calls and tests all these functions.

Question 4:

Implement the function:

```
void oddsKeepEvensFlip(int arr[], int arrSize)
```

This function gets an array of integers `arr` and its logical size `arrSize`.

When called, it should **reorder** (in-place) the elements of `arr` so that:

1. All the odd numbers come before all the even numbers
2. The odd numbers will keep their original relative order
3. The even numbers will be placed in a reversed order (relative to their original order).

For example, if `arr = [5, 2, 11, 7, 6, 4]`,

after calling `oddsKeepEvensFlip(arr, 6)`, `arr` will be:

```
[5, 11, 7, 4, 6, 2]
```

Implementation requirements:

1. Your function should run in **linear time**. That is, if there are n items in `arr`, calling `oddsKeepEvensFlip(arr, n)` will run in $\theta(n)$.
2. Write a `main()` program that tests this function..